第四十五章：Redis高性能数据库

**一、Redis概述；**

**二、应用场景分析；**

**三、Redis的数据回收策略；**

**四、案例：搭建Redis高性能数据库；**

**五、扩展：Nginx+Tomcat+Mysql+Redis+实现session回话共享；**

**一、Redis概述：**

**概述：**redis是一个可持久化的单进程单线程key-value类型高速内存缓存系统，和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(列表)、set(集合)、zset(sorted set --有序集合)和hashs（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作。

与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会定期通过异步操作把数据库数据flush到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知性能最快的Key-Value DB，并且在此基础上实现了master-slave(主从)同步,当前 Redis的应用已经非常广泛，国内像新浪、淘宝，国外像 Flickr、Github等均在使用Redis的缓存服务。官网地址：https://redis.io/ ，目前由redis项目的开发和维护由vmware厂商负责提供支持；

**优点：**

1.性能极高--Redis能支持超过 100K+ 每秒的读写频率。

2.丰富的数据类型--Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

3.原子性--Redis的所有操作都是原子性（即所有操作要么成功要么失败不执行），同时Redis还支持对几个操作全并后的原子性执行。

4.Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

**二、应用场景分析：**

**MySql+Memcached架构的问题：**实际MySQL是适合进行海量数据存储的，通过Memcached将热点数据加载到cache，加速访问，很多公司都曾经使用过这样的架构，但随着业务数据量的不断增加，和访问量的持续增长，我们遇到了很多问题：

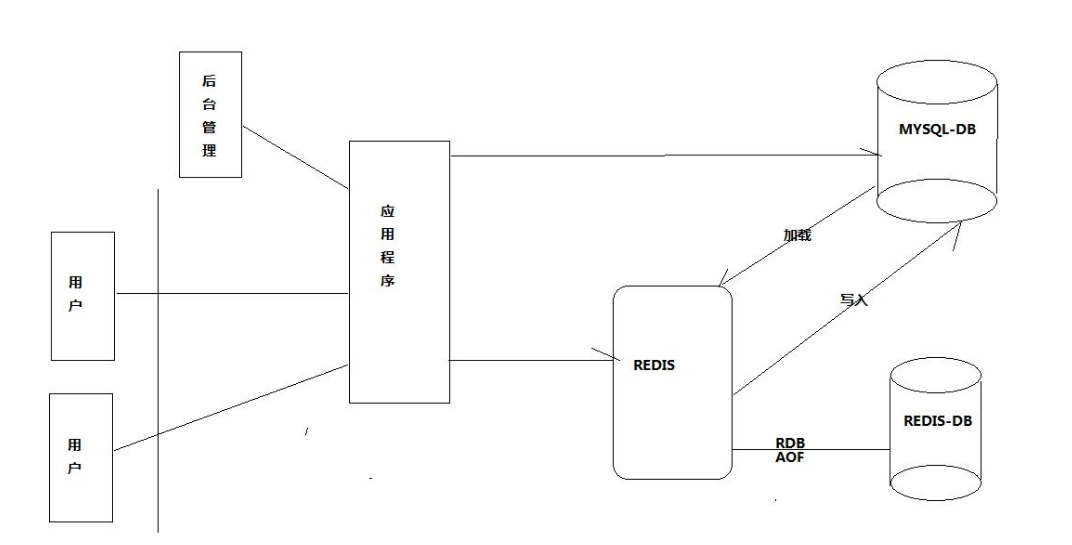
1.MySQL需要不断进行拆库拆表，Memcached也需不断跟着扩容，扩容和维护工作占据大量开发时间。

2.Memcached与MySQL数据库数据一致性问题。

3.Memcached数据命中率低或down机，大量访问直接穿透到DB，MySQL无法支撑。

4.跨机房cache同步问题。

**Redis的应用场景：**



Redis更像一个加强版的Memcached，那么何时使用Memcached,何时使用Redis呢?

1.Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。  
2.Redis支持数据的备份，即master-slave模式的数据备份。  
3.Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

4.Memcached：动态系统中减轻数据库负载，提升性能；做缓存，适合多读少写，大数据量的情况（如人人网大量查询用户信息、好友信息、文章信息等）。

 Redis：适用于对读写效率要求都很高，数据处理业务复杂和对安全性要求较高的系统（如新浪微博的计数和微博发布部分系统，对数据安全性、读写要求都很高）。

**三、Redis的数据回收策略：**

**Redis的回收策略：**

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰；

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰；

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰；

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰；

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰；

no-enviction（驱逐）：禁止驱逐数据；

**四、案例：搭建Redis高性能数据库：**

**案例环境：**

|  |  |  |  |
| --- | --- | --- | --- |
| 系统 | 主机名 | IP地址 | 软件 |
| Centos 7.4 | redis.linuxfan.cn | 192.168.100.101 | redis-4.0.9.tar.gz  phpredis-4.0.2.tar.gz |

**实验步骤：**

* 安装并配置Redis服务；
* 调整Redis服务的配置；
* 结合PHP测试Redis服务；
* Redis中操作之String；
* Redis中操作之list；
* Redis中操作之set；
* Redis中操作之zset；
* Redis中操作之hash；
* Redis中操作之其他操作；
* **安装并配置Redis服务；**

[root@redis ~]# wget http://download.redis.io/releases/redis-4.0.9.tar.gz

[root@redis ~]# tar zxvf redis-4.0.9.tar.gz

[root@redis ~]# cd redis-4.0.9

[root@redis redis-4.0.9]# make

[root@redis redis-4.0.9]# echo $?

[root@redis redis-4.0.9]# cd

[root@redis ~]# mkdir -p /usr/local/redis

[root@redis ~]# cp /root/redis-4.0.9/src/redis-server /usr/local/redis/ ##服务端程序

[root@redis ~]# cp /root/redis-4.0.9/src/redis-cli /usr/local/redis/ ##客户端程序

[root@redis ~]# cp /root/redis-4.0.9/redis.conf /usr/local/redis/ ##主配置文件

[root@redis ~]# ls /usr/local/redis/

redis-cli redis.conf redis-server

[root@redis ~]# sed -i '/^bind 127.0.0.1$/s/127.0.0.1/192.168.100.101/g' /usr/local/redis/redis.conf

[root@redis ~]# sed -i '/protected-mode/s/yes/no/g' /usr/local/redis/redis.conf ##关闭redis的保护模式

[root@redis ~]# sed -i '/daemonize/s/no/yes/g' /usr/local/redis/redis.conf ##开启redis的后台守护进程模式

[root@redis ~]# sed -i '/requirepass/s/foobared/123123/g' /usr/local/redis/redis.conf ##设置redis的密码为123123

[root@redis ~]# sed -i '/requirepass 123123/s/^#//g' /usr/local/redis/redis.conf ##开启redis的密码

附：redis.conf主配置文件详解

daemonize yes #以后台daemon方式运行redis

pidfile "/var/run/redis.pid" #redis以后台运行，默认pid文件路径/var/run/redis.pid

port 6379 #默认端口

bind 127.0.0.1 #默认绑定本机所有ip地址，为了安全，可以只监听内网ip

timeout 300 #客户端超时设置，单位为秒

loglevel verbose #设置日志级别，支持四个级别：debug、notice、verbose、warning

logfile stdout #日志记录方式，默认为标准输出，logs不写文件，输出到空设备/deb/null

logfile "/usr/local/redis/var/redis.log" #可以指定日志文件路径

databases 16 #开启数据库的数量

save 900 1

save 300 10

save 60 10000

创建本地数据库快照，格式：save \* \*

900秒内，执行1次写操作后触发快照

300秒内，执行10次写操作

60秒内，执行10000次写操作

rdbcompression yes #启用数据库lzf压缩，也可以设置为no

dbfilename dump.rdb #本地快照数据库名称

dir "/usr/local/redis/var/" #本地快照数据库存放目录

requirepass 123456 #设置redis数据库连接密码

maxclients 10000 #同一时间最大客户端连接数，0为无限制

maxmemory 1024MB #设定redis最大使用内存，值要小于物理内存，必须设置

appendonly yes #开启日志记录，相当于MySQL的binlog

appendfilename "appendonly.aof" #日志文件名，注意：不是目录路径

appendfsync everysec #设置日志同步的频率，每秒执行同步，还有两个参数always、no一般设置为everysec，相当于MySQL事物日志的写方式

Slaveof 设置数据库为其他数据库的从数据库

Masterauth 主数据库连接需要的密码验证

vm-enabled 是否开启虚拟内存支持 （vm开头的参数都是配置虚拟内存的）

vm-swap-file 设置虚拟内存的交换文件路径

vm-max-memory 设置redis使用的最大物理内存大小

vm-page-size 设置虚拟内存的页大小

vm-pages 设置交换文件的总的page数量

vm-max-threads 设置使用swap存储同时使用的线程数量，通常设置值为核心数相同，如果设置为0，则会以串行方式，对数据的完整性有着极大的保证

Glueoutputbuf 把小的输出缓存存放在一起

hash-max-zipmap-entries 设置hash的临界值

Activerehashing 重新hash

[root@redis ~]# cat <<END >>/etc/init.d/redis

#!/bin/sh

# chkconfig: 2345 80 90

# description: Start and Stop redis

#PATH=/usr/local/bin:/sbin:/usr/bin:/bin

REDISPORT=6379

EXEC=/usr/local/redis/redis-server

REDIS\_CLI=/usr/local/redis/redis-cli

PIDFILE=/var/run/redis\_6379.pid

CONF="/usr/local/redis/redis.conf"

AUTH="123123"

LISTEN\_IP=\$(netstat -utpln |grep redis-server |awk '{print \$4}'|awk -F':' '{print \$1}')

case "\$1" in

start)

if [ -f \$PIDFILE ]

then

echo "\$PIDFILE exists, process is already running or crashed"

else

echo "Starting Redis server..."

\$EXEC \$CONF

fi

if [ "\$?"="0" ]

then

echo "Redis is running..."

fi

;;

stop)

if [ ! -f \$PIDFILE ]

then

echo "\$PIDFILE does not exist, process is not running"

else

PID=\$(cat \$PIDFILE)

echo "Stopping ..."

\$REDIS\_CLI -h \$LISTEN\_IP -p \$REDISPORT -a \$AUTH SHUTDOWN

while [ -x \${PIDFILE} ]

do

echo "Waiting for Redis to shutdown ..."

sleep 1

done

echo "Redis stopped"

fi

;;

restart|force-reload)

\${0} stop

\${0} start

;;

\*)

echo "Usage: /etc/init.d/redis {start|stop|restart|force-reload}" >&2

exit 1

esac

END

[root@redis ~]# chmod 755 /etc/init.d/redis

[root@redis ~]# chkconfig --add redis

[root@redis ~]# /etc/init.d/redis start

Starting Redis server...

4390:C 04 May 02:16:45.232 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo

4390:C 04 May 02:16:45.232 # Redis version=4.0.9, bits=64, commit=00000000, modified=0, pid=4390, just started

4390:C 04 May 02:16:45.232 # Configuration loaded

Redis is running...

[root@redis ~]# netstat -utpln |grep redis

tcp 0 192.168.100.101:6379 0.0.0.0:\* LISTEN 4204/redis-server \*

* **调整Redis服务的配置：**

[root@redis ~]# cp /root/redis-4.0.9/src/redis-benchmark /usr/local/redis/

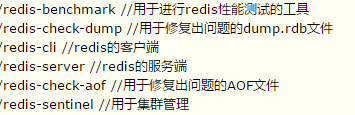
[root@redis ~]# cp /root/redis-4.0.9/src/redis-check-rdb /usr/local/redis/

[root@redis ~]# cp /root/redis-4.0.9/src/redis-check-aof /usr/local/redis/

[root@redis ~]# cp /root/redis-4.0.9/src/redis-sentinel /usr/local/redis/

[root@redis ~]# ls /usr/local/redis/

dump.rdb redis-benchmark redis-cli redis.conf redis-server



[root@redis ~]# ln -s /usr/local/redis/redis-cli /usr/bin/redis

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> set name lwh

OK

192.168.100.101:6379> get name

"lwh"

192.168.100.101:6379> exit

[root@redis ~]# ln -s /usr/local/redis/redis-benchmark /usr/bin/redis-benchmark

[root@redis ~]# redis-benchmark -h 192.168.100.101 -p 6379 -c 1000 -n 10000

* **结合PHP测试Redis服务；**

[root@redis ~]# yum -y install httpd php php-redis php-devel

[root@redis ~]# php -v

PHP 5.4.16 (cli) (built: Mar 7 2018 13:34:47)

Copyright (c) 1997-2013 The PHP Group

Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies

[root@redis ~]# wget https://codeload.github.com/phpredis/phpredis/tar.gz/4.0.2

[root@redis ~]# tar zxvf phpredis-4.0.2.tar.gz

[root@redis ~]# cd phpredis-4.0.2

[root@redis phpredis-4.0.2]# /usr/bin/phpize

Configuring for:

PHP Api Version: 20100412

Zend Module Api No: 20100525

Zend Extension Api No: 220100525

[root@redis phpredis-4.0.2]# ./configure --with-php-config=/usr/bin/php-config

[root@redis phpredis-4.0.2]# make && make install

[root@redis phpredis-4.0.2]# echo $?

0

[root@redis phpredis-4.0.2]# cd

[root@redis ~]# echo -e "extension\_dir = \"/usr/lib64/php/modules/\"\nextension = redis.so" >>/etc/php.ini

[root@redis ~]# systemctl start httpd

[root@redis ~]# netstat -utpln |grep 80

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 19008/httpd

[root@redis ~]# cat <<END >>/var/www/html/index.php

<?php

phpinfo();

?>

END

[root@redis ~]# cat <<END >>/var/www/html/test.php

<?php

\$redis = new Redis();

\$redis->connect('192.168.100.101',6379);

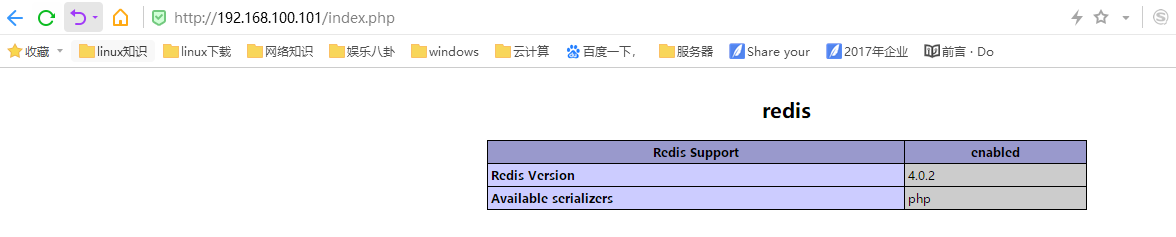
\$redis->auth('123123');

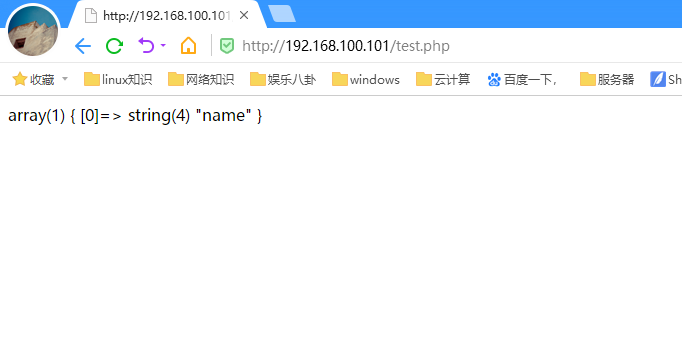
\$keys = \$redis->keys("\*");

var\_dump(\$keys);

?>

END





* **Redis中操作之String -- 字符串（注意：redis中的语句可以tab补全）；**

**概述：String是简单的 key-value 键值对，value 不仅可以是 String，也可以是数字。String在redis内部存储默认就是一个字符串，被redisObject所引用，当遇到incr,decr等操作时会转成数值型进行计算，此时redisObject的encoding字段为int。**

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> set hello world

OK

192.168.100.101:6379> get hello

"world"

192.168.100.101:6379> EXISTS hello ##验证key是否存在

(integer) 1

192.168.100.101:6379> type hello

string

192.168.100.101:6379> substr hello 1 2 ##查看key的第1到2两个字符

"or"

192.168.100.101:6379> append hello ! ##在key后添加字符!

(integer) 6

192.168.100.101:6379> get hello

"world!"

192.168.100.101:6379> set haha heihei

OK

192.168.100.101:6379> keys h\* ##查看以h开头的Key

1) "haha"

2) "hello"

192.168.100.101:6379> set name xiaoming

OK

192.168.100.101:6379> keys \*

1) "haha"

2) "hello"

3) "name"

192.168.100.101:6379> RANDOMKEY ##随机返回一个Key

"name"

192.168.100.101:6379> keys \*

1) "haha"

2) "hello"

3) "name"

192.168.100.101:6379> RANDOMKEY

"haha"

192.168.100.101:6379> RENAME haha hehe

OK

192.168.100.101:6379> keys \*

1) "hehe"

2) "name"

3) "hello"

192.168.100.101:6379> DEL hehe

(integer) 1

192.168.100.101:6379> keys \*

1) "name"

2) "hello"

192.168.100.101:6379> get name

"xiaoming"

192.168.100.101:6379> set name xiaohong

OK

192.168.100.101:6379> get name

"xiaohong"

192.168.100.101:6379> expire name 10 ##设置KEY值的超时时间

(integer) 1

192.168.100.101:6379> ttl name ##查看KEY值当前剩余的超时时间

(integer) 4

192.168.100.101:6379> get name

"xiaohong"

192.168.100.101:6379> get name

(nil)

192.168.100.101:6379> keys \*

1) "hello"

* **Redis中操作之list -- 列表；**

**概述：Redis列表是简单的字符串列表，可以类比到C++中的std::list，简单的说就是一个列表或者说是一个队列。可以从头部或尾部向Redis列表添加元素。列表的最大长度为2^32 - 1，也即每个列表支持超过40亿个元素。Redis list的实现为一个双向列表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。**

**应用场景：Redis list的应用场景非常多，也是Redis最重要的数据结构之一，比如twitter的关注列表、粉丝列表等都可以用Redis的list结构来实现，再比如有的应用使用Redis的list类型实现一个简单的轻量级消息队列，生产者push，消费者pop/bpop。**

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> rpush list1 1 ##创建列表并在尾部插入元素

(integer) 1

192.168.100.101:6379> rpush list1 2

(integer) 2

192.168.100.101:6379> lrange list1 0 1 ##验证列表中的第0个元素到第1个元素

1) "1"

2) "2"

192.168.100.101:6379> lpush list1 0 ##在列表头部插入元素

(integer) 3

192.168.100.101:6379> lrange list1 0 1

1) "0"

2) "1"

192.168.100.101:6379> lrange list1 0 2 ##验证列表中的第0个元素到第2个元素

1) "0"

2) "1"

3) "2"

192.168.100.101:6379> llen list1 ##查看列表中的元素的数量

(integer) 3

192.168.100.101:6379> lindex list1 1 ##定位到列表中的第1个元素

"1"

192.168.100.101:6379> ltrim list1 1 2 ##截取列表中的第1个元素到第2个元素

OK

192.168.100.101:6379> lrange list1 0 10 ##查看列表中第0个元素到第10个元素

1) "1"

2) "2"

192.168.100.101:6379> lset list1 1 haha ##更改列表中的第1个元素值为haha

OK

192.168.100.101:6379> lrange list1 0 10

1) "1"

2) "haha"

192.168.100.101:6379> lset list1 2 haha ##验证更改时的元素位置不能大于现有元素的数量

(error) ERR index out of range

192.168.100.101:6379> lrange list1 0 10

1) "1"

2) "haha"

192.168.100.101:6379> rpush list1 haha ##在列表尾部插入新的元素

(integer) 3

192.168.100.101:6379> lrange list1 0 10

1) "1"

2) "haha"

3) "haha"

192.168.100.101:6379> lrem list1 2 haha ##删除列表中两个值为haha的元素

(integer) 2

192.168.100.101:6379> lrange list1 0 10

1) "1"

192.168.100.101:6379> rpush list1 haha

(integer) 2

192.168.100.101:6379> rpush list1 haha

(integer) 3

192.168.100.101:6379> rpush list1 haha

(integer) 4

192.168.100.101:6379> rpush list1 haha

(integer) 5

192.168.100.101:6379> lrange list1 0 10

1) "1"

2) "haha"

3) "haha"

4) "haha"

5) "haha"

192.168.100.101:6379> lpop list1 ##删除列表内开头的第一个元素

"1"

192.168.100.101:6379> lrange list1 0 10

1) "haha"

2) "haha"

3) "haha"

4) "haha"

192.168.100.101:6379> lpop list1

"haha"

192.168.100.101:6379> lrange list1 0 10

1) "haha"

2) "haha"

3) "haha"

* **Redis中操作之set -- 无序集合；**

**概述：可以理解为一堆值不重复的列表，类似数学领域中的集合概念，且Redis也提供了针对集合的求交集、并集、差集等操作。set 的内部实现是一个 value永远为null的HashMap，实际就是通过计算hash的方式来快速排重的，这也是set能提供判断一个成员是否在集合内的原因。**

**应用场景：Redis set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。又或者在微博应用中，每个用户关注的人存在一个集合中，就很容易实现求两个人的共同好友功能。**

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> sadd set1 0 ##创建set1并赋值0

(integer) 1

192.168.100.101:6379> sadd set1 1 ##在set1后追加值1

(integer) 1

192.168.100.101:6379> smembers set1 ##显示set1的所有值

1) "0"

2) "1"

192.168.100.101:6379> scard set1 ##显示set1的基数

(integer) 2

192.168.100.101:6379> sismember set1 0 ##显示set1中是否包含值为0的元素

(integer) 1

192.168.100.101:6379> srandmember set1 ##随机返回set1中的的元素值

"0"

192.168.100.101:6379> sadd set2 0 ##创建set2并添加元素

(integer) 1

192.168.100.101:6379> sadd set2 2

(integer) 1

192.168.100.101:6379> sinter set1 set2 ##过滤set1与set2的交集

1) "0"

192.168.100.101:6379> sinterstore set3 set1 set2 ##将set1与set2的交集保存到set3

(integer) 1

192.168.100.101:6379> smembers set3

1) "0"

192.168.100.101:6379> sunion set1 set2 ##过滤set1与set2的并集

1) "0"

2) "1"

3) "2"

192.168.100.101:6379> sdiff set1 set2 ##相对与set2来讲，set1的差集

1) "1"

192.168.100.101:6379> sdiff set2 set1 ##相对比set1来讲，set2的差集

1) "2"

192.168.100.101:6379> keys \* ##查看已经存在的各种数据类型的键

1) "set1"

2) "hello"

3) "list"

4) "list1"

5) "set2"

* **Redis中操作之zset -- 有序集合；**

**概述：Redis有序集合类似Redis无序集合，不同的是增加了一个功能，即集合是有序的。一个有序集合的每个成员带有分数，用于进行排序。Redis有序集合添加、删除和测试的时间复杂度均为O(1)(固定时间，无论里面包含的元素集合的数量)。列表的最大长度为2^32- 1元素(4294967295，超过40亿每个元素的集合)。**

**Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。**

**使用场景：Redis sorted set的使用场景与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。当你需要一个有序的并且不重复的集合列表，那么可以选择sorted set数据结构；**

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> zadd zset1 1 baidu.com ##创建集合zset1，添加元素baidu.com，并且设置元素的score值（有序集合的序列号）为1

(integer) 1

192.168.100.101:6379> zadd zset1 2 sina.com

(integer) 1

192.168.100.101:6379> zadd zset1 3 qq.com

(integer) 1

192.168.100.101:6379> zrange zset1 0 2 ##查看集合zset1中的元素

1) "baidu.com"

2) "sina.com"

3) "qq.com"

192.168.100.101:6379> zcard zset1 ##查看集合zset1中的基数

(integer) 3

192.168.100.101:6379> zscore zset1 baidu.com ##验证元素baidu.com在集合zset1中的score值

"1"

192.168.100.101:6379> zscore zset1 qq.com ##验证元素qq.com在集合zset1中的score值

"3"

192.168.100.101:6379> zrevrange zset1 0 1 ##倒序查找集合中的元素，

1) "qq.com"

2) "sina.com"

192.168.100.101:6379> zrem zset1 qq.com ##删除集合中的元素qq.com

(integer) 1

192.168.100.101:6379> zrange zset1 0 5 ##验证集合中的所有元素

1) "baidu.com"

2) "sina.com"

192.168.100.101:6379> zincrby zset1 5 taobao.com ##设置该元素score值为5，如若已经存在score值为5的元素，那么则按照数字0-9，字母a-z进行自动排序；

"5"

192.168.100.101:6379> zrange zset1 0 5 ##验证集合中的元素

1) "baidu.com"

2) "sina.com"

3) "taobao.com"

192.168.100.101:6379> zrange zset1 0 5 withscores ##查看集合中的元素值并且输出其本身的score值

1) "baidu.com"

2) "1"

3) "sina.com"

4) "2"

5) "taobao.com"

6) "5"

192.168.100.101:6379> zincrby zset1 10 haha.com ##添加新元素haha.com并指定其本身score值

"10"

192.168.100.101:6379> zrange zset1 0 5 withscores ##验证元素的排序

1) "baidu.com"

2) "1"

3) "sina.com"

4) "2"

5) "taobao.com"

6) "5"

7) "haha.com"

8) "10"

192.168.100.101:6379> zincrby zset1 15 baidu.com ##如若新添加的元素已经存在，那么会在原有score值的基础上增加指定得score值

"16"

192.168.100.101:6379> zrange zset1 0 5 withscores ##验证元素baidu.com的score值变为16，增加了15，并排序到集合的末尾

1) "sina.com"

2) "2"

3) "taobao.com"

4) "5"

5) "haha.com"

6) "10"

7) "baidu.com"

8) "16"

**注：此外, 还有zrevrank, zrevrange, zrangebyscore, zremrangebyrank, zramrangebyscore, zinterstore/zunionstore等操作**

* **Redis中操作之hash ；**

**概述：类似C#中的dict类型或者C++中的hash\_map类型。Redis Hash对应Value内部实际就是一个HashMap，这个Hash的成员比较少时Redis为了节省内存会采用类似一堆数组的方式来紧凑存储，而不会采用真正的HashMap结构，对应的value redisObject的encoding为zipmap，当成员数量增大时会自动转成真正的HashMap,此时encoding为ht。**

**应用场景：假设有多个用户及对应的用户信息，可以用来存储以用户ID为key，将用户信息序列化为比如json格式做为value进行保存。**

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> hset hash1 key1 value1 ##设置键值hash1，并且指定内部key与value对应

(integer) 1

192.168.100.101:6379> hget hash1 key1 ##查看hash1中的key1的值

"value1"

192.168.100.101:6379> hexists hash1 key1 ##检查hash1中是否有key1

(integer) 1

192.168.100.101:6379> hset hash1 key2 value2 ##在hash1中创建key2

(integer) 1

192.168.100.101:6379> hlen hash1 ##查看hash1中键值的基数

(integer) 2

192.168.100.101:6379> hkeys hash1 ##查看hash1中具体包含的键

1) "key1"

2) "key2"

192.168.100.101:6379> hvals hash1 ##查看hash1中具体包含的值

1) "value1"

2) "value2"

192.168.100.101:6379> hmget hash1 key1 key2 ##通过明确指定hash1中的key，去查看值是什么

1) "value1"

2) "value2"

192.168.100.101:6379> hgetall hash1 ##查看hash1中包含的所有键和值的对应

1) "key1"

2) "value1"

3) "key2"

4) "value2"

192.168.100.101:6379> hset hash1 key4 10 ##添加新键key4，值为整数10

(integer) 1

192.168.100.101:6379> hincrby hash1 key4 15 ##将key4执行计算加10（仅限整数）

(integer) 25

192.168.100.101:6379> hmset hash1 key5 value5 key6 value6 key7 value7 ##批量添加键值对

OK

192.168.100.101:6379> hgetall hash1 ##验证批量添加的键值对

1) "key1"

2) "value1"

3) "key2"

4) "value2"

5) "key3"

6) "value3"

7) "key4"

8) "25"

9) "key5"

10) "value5"

11) "key6"

12) "value6"

13) "key7"

14) "value7"

* **Redis中操作之其他操作；**

[root@redis ~]# redis -h 192.168.100.101 -p 6379 -a 123123

192.168.100.101:6379> dbsize ##查看所有key的数目   
192.168.100.101:6379> flushdb ##删除当前选择数据库中的所有key   
192.168.100.101:6379> flushall ##删除所有数据库中的所有key   
192.168.100.101:6379> save ##将数据同步保存到磁盘   
192.168.100.101:6379> bgsave ##异步保存   
192.168.100.101:6379> lastsave ##上次成功保存到磁盘的Unix时间戳   
192.168.100.101:6379> info ##查询server信息   
192.168.100.101:6379> slaveof ##改变复制策略设置

**五、扩展：Nginx+Tomcat+Mysql+Redis+实现session回话共享；**

http://blog.51cto.com/yw666/1910451